

BEDFORD OPERATIONS

MTR-3449

ARPA Order Nos. 3338, 2223

(1)

~~SECRET~~

# Design of a Message Processing System for a Multilevel Secure Environment

S. R. Ames, Jr., MITRE

D. R. Oestreicher, USC/Information Sciences Institute

JUNE 1978

APPROVED FOR PUBLIC RELEASE  
DISTRIBUTION UNLIMITED

DTIC FILE COPY

DTIC  
ELECTE  
NOV 28 1983

B

MITRE

83 11 28 186

SECRET

AD-A234954

MITRE Technical Report

MTR-3449

ARPA Order Nos. 3338, 2223

# Design of a Message Processing System for a Multilevel Secure Environment

S. R. Ames, Jr., MITRE

D. R. Oestreicher, USC/Information Sciences Institute

JUNE 1978

CONTRACT SPONSOR  
CONTRACT NO.  
PROJECT NO.  
DEPT.

DARPA  
F19628-78-C-0001  
8010  
D-75

Presented at 1978 National Computer Conference.

The views and conclusions contained in this paper are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied of the Defense Advanced Research Projects Agency or the United States Government.

THE  
**MITRE**  
CORPORATION  
BEDFORD, MASSACHUSETTS

Approved for public release; distribution unlimited.

# ABSTRACT

Is it possible to build a message processing system that not only provides the user with a clean usable interface but is also multi-level secure? To answer that question the Defense Advanced Research Projects Agency and the Navy specified both of these as requirements in their test of a military message processing system. We believe that the SIGMA message service built by Information Sciences Institute provides both.

This paper presents the security design of SIGMA, which includes: a description of the user interface to security, a description of how SIGMA provides that interface securely, and a description of what a security kernel must provide in order to support SIGMA efficiently.

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	



## TABLE OF CONTENTS

<u>Section</u>	<u>Page</u>
LIST OF ILLUSTRATIONS	vi
I INTRODUCTION	1
II THE SIGMA MESSAGE PROCESSING SYSTEM	2
III THE KERNEL APPROACH TO MULTILEVEL SECURITY	3
IV SECURITY PROBLEMS OF MESSAGE PROCESSING SYSTEMS	5
V SECURE MESSAGE PROCESSING SYSTEM ARCHITECTURE	6
OVERVIEW OF THE DESIGN	6
MULTILEVEL TERMINAL	8
FORM OF COMMAND INPUT	8
TRUSTED PROCESS FUNCTIONS	10
Change Classification	11
Message Release	11
Command Completion Signals	11
Entry Lists	12
VI KERNEL REQUIREMENTS	13
MULTILEVEL TERMINAL CERTIFICATION	13
TERMINAL MULTIPLEXOR	13
PROCESS STRUCTURE	14
INTERPROCESS COMMUNICATION	15
FILE SYSTEM	15
SYSTEM INTEGRITY CONTROLS	15
VII SUMMARY	16
REFERENCES	17
DISTRIBUTION LIST	19

## LIST OF ILLUSTRATIONS

<u>Figure</u>		<u>Page</u>
1	Sigma Process Structure	7
2	Multi Level Terminal	9

## SECTION I

### INTRODUCTION

The Department of Defense Advanced Research Projects Agency (DARPA) and the Navy are currently conducting an experiment to evaluate the operational use and organizational impact of a computer-aided message handling system. An important aspect of this experiment was to design a system with sufficient security controls to enable it to process messages at multiple levels of classification. An equally important aspect of the experiment was for the system to exhibit a rich user interface that was judged easy to learn and use. Herein we present the security aspects of the design for the SIGMA Message Processing System, the system chosen for the experiment. (1) *e*

*In Section 2,* <sup>3</sup>  
~~In the following~~ section, a description of the SIGMA Message Processing System is given. Section ~~III~~ provides background and discusses the kernel approach to multilevel security. *The authors* We describe in Section ~~IV~~ several security problems encountered in the design. Section V presents the design of the SIGMA message service. The additional features that the kernel must provide to support SIGMA efficiently are documented in Section VI. Finally, a summary is provided to highlight the paper's main points.

*5* *6*

(1) The SIGMA message service is one of three services developed during this experiment. The other two are the HERMES system built by Bolt, Beranek and Newman, Inc. [1] and DMS built by Massachusetts Institute of Technology [2].

## SECTION II

### THE SIGMA MESSAGE PROCESSING SYSTEM

Information Sciences Institute of the University of Southern California developed SIGMA specifically to meet the message handling needs of a military command. SIGMA is a secure interactive message handling system providing computer-aided message handling services for the receipt, filing, retrieval, creation, and coordination of military (AUTODIN) messages. We consider it secure in that it presents an interface to the user constrained to abide by the DoD security policy. It is an interactive system since all user-system communications occur via an on-line terminal with a CRT display. Finally, it is a message handling system because it supports the typical message processing functions needed by any formal organization's operation.

SIGMA supports the full cycle for processing incoming and outgoing messages in a military operation. It provides flexible filing capabilities for on-line storage of all messages. Easy access to messages and files is provided by selective search and retrieval functions. Incoming AUTODIN messages move through the system by informal forwarding or by formal action assignment. Outgoing messages are processed by a set of functions supporting message creation, editing, coordination, release, and post-transmission comeback copies.

SIGMA operates on a DEC PDP-10 computer with the TENEX operating system. AUTODIN messages enter through the local AUTODIN message exchange. SIGMA distributes these messages to all pertinent addressees on the system where each user can access them through his SIGMA display terminal.

### SECTION III

#### THE KERNEL APPROACH TO MULTILEVEL SECURITY

The need to process multiple levels of classified data led the Air Force Electronic Systems Division to sponsor several research and development efforts to build an operating system that could satisfy by technical verification that DoD security requirements had been met [3]. Many of the results of the ESD work have been borrowed in the design of SIGMA, specifically adherence to a mathematical model based on the concept of a reference monitor--an abstract mechanism that controls the flow of information within a computer system by mediating every attempt by a subject (active system element) to access an object (information container).(2) [4] The hardware/software mechanism that implements the reference monitor is called a security kernel. The kernel uses the rules of the mathematical model [5] [6] as a specific policy for mediating access requests. This incorporation of policy into the kernel allows for a proof that verifies that the kernel correctly applies the policy to the information it protects. [7] [8]

The mathematical model establishes an "inductive nature" of security by demonstrating that security is preserved from one state to another. Security is defined with two rules: the simple security condition and the \*-property [9]. The former states that a subject (active entity) cannot observe the contents of an object (information container) unless its security level is greater than or equal to the security level of the object.(3) The latter further restricts possible access by stipulating that a subject may only modify an object if that object's security level is greater than or equal to the security level of the subject.

---

(2) In a computer system, subjects are users and processors, and objects include programs, data files, and peripheral devices.

(3) Currently the security levels used in SIGMA are only the four standard DoD classifications, i.e., Unclassified, Confidential, Secret, and Top Secret.

The purpose of the simple security condition is to prohibit users from obtaining data that they are not entitled to see. The \*-property is designed to prohibit a program operating on behalf of a user from reducing the classification of any information.

When a user is given a clearance, he is charged with responsibility for maintaining the classification of classified information. A computer utility cannot necessarily be given this same trust. This is due to: the amount of information that may be compromised; the speed with which the compromise may occur; and the difficulty in detecting or apprehending the violating program. By enforcing the \*-property on computer programs, a program will not be able to either accidentally or maliciously compromise information. Designers of computer utilities constrained by the \*-property must ensure that \*-property enforcement does not unnecessarily restrict the capabilities of the user.

The enforcement of the \*-property allows us to reduce the volume of code that needs to be trusted to a central section of the operating system. This central section of the operating system is the software component of the security kernel. To provide security, a kernel must 1) mediate every access by a subject to an object, 2) be protected from unauthorized modification, and 3) correctly perform its functions. A kernel satisfies the first requirement by creating an environment within which all non-kernel software is constrained to operate and by maintaining control over it.

The requirement to protect against unauthorized modification is satisfied by isolating the security kernel software in one or more protection domains, for example, by a ring mechanism [10]. Finally, the requirement that the kernel correctly perform its functions is satisfied by using a formal methodology. A suitable methodology was introduced by Bell and Burke [7]. It includes: 1) a proof that the kernel behavior enforces the desired policy [11]; and 2) a proof that the kernel is correctly implemented with respect to the description of its behavior used in the first step [12].

We designed SIGMA with security kernel technology in mind. However, due to the absence of a kernel on the PDP-10 (the machine we used), the current implementation was done without a kernel. We have rigorously scrutinized the SIGMA design to ensure that the user interface provided would remain unchanged should SIGMA be reimplemented on a security kernel. In addition, the security primitives have been evaluated to ensure that their usefulness warrants their being included in a kernel.

## SECTION IV

### SECURITY PROBLEMS OF MESSAGE PROCESSING SYSTEMS

A kernel supporting the current mathematical model of the DoD security policy is well suited for certain environments, such as a programming environment in which users operate at a single security level for long periods of time. [13] A message processing environment presents several problems not found in previous environments, including 1) the dynamic nature of the user's "working security level"; 2) the desire to present to the user information at more than a single security level; 3) the desire to accurately inform the user of the security level of all information he is reading or writing; and 4) the ability of users to extract text information and place it in a message of a lower classification than the source.

The user's "working security level" in a message system environment is considerably more dynamic than in the programming environment. Each time that a user performs an action on a different message, his working security level may have to change; for example, a user reading a Secret message may generate an Unclassified reply. While we could require the user to process messages at a single security level at a time, the resulting user interface would be clearly unacceptable to the user. [14]

To deal with these problems a new approach is needed that includes: a terminal that will allow users to process information at more than one security level at a time; and trusted processes that are able to violate the security rules in a controlled manner. The next section describes the security architecture of the SIGMA message processing system.

## SECTION V

### SECURE MESSAGE PROCESSING SYSTEM ARCHITECTURE

The SIGMA security design has two goals: to produce a certifiably secure service, and to present the user with an agreeable user interface. In many situations these goals are at cross-purposes. Our general approach has been to present the user with a true picture of what is happening, maintain the user's data at the proper level (or higher if this is not possible), and make it convenient for the user to do the right thing. [14]

#### OVERVIEW OF THE DESIGN

When using SIGMA, a user is actually interacting with a collection of up to five processes (see Figure 1). These are the trusted process, an unclassified control process, and one process for each classified level that the user/terminal is cleared to operate. Each process (except the trusted process) can write data only at its own level and can read data at its level or lower.

SIGMA attempts to be as helpful to the user as it can, organizing the user's session and cleaning up the user's state (current context) as necessary. The service also attempts to understand the user's current context and conform its behavior to the situation. For this reason the context information must be available to all user processes; thus, it must be unclassified.

The user's state in SIGMA is divided into two parts. The first part contains the current list of objects being accessed and functions being performed by the user. This portion of the state is maintained at the unclassified level by the unclassified control process. The second part contains the current list of message entries (from the open message file) in which the user has expressed an interest. The entry list information is potentially classified at the level of the file and is thus maintained at this classification level by the appropriate classified process.

This dichotomy of state is reflected directly into the security design. Commands are divided into those which access the unclassified state (unclassified commands) and those which access the entry list (classified commands). The latter group includes both commands that use the entry list for input and those that allow the user to enter classified information as part of the command.

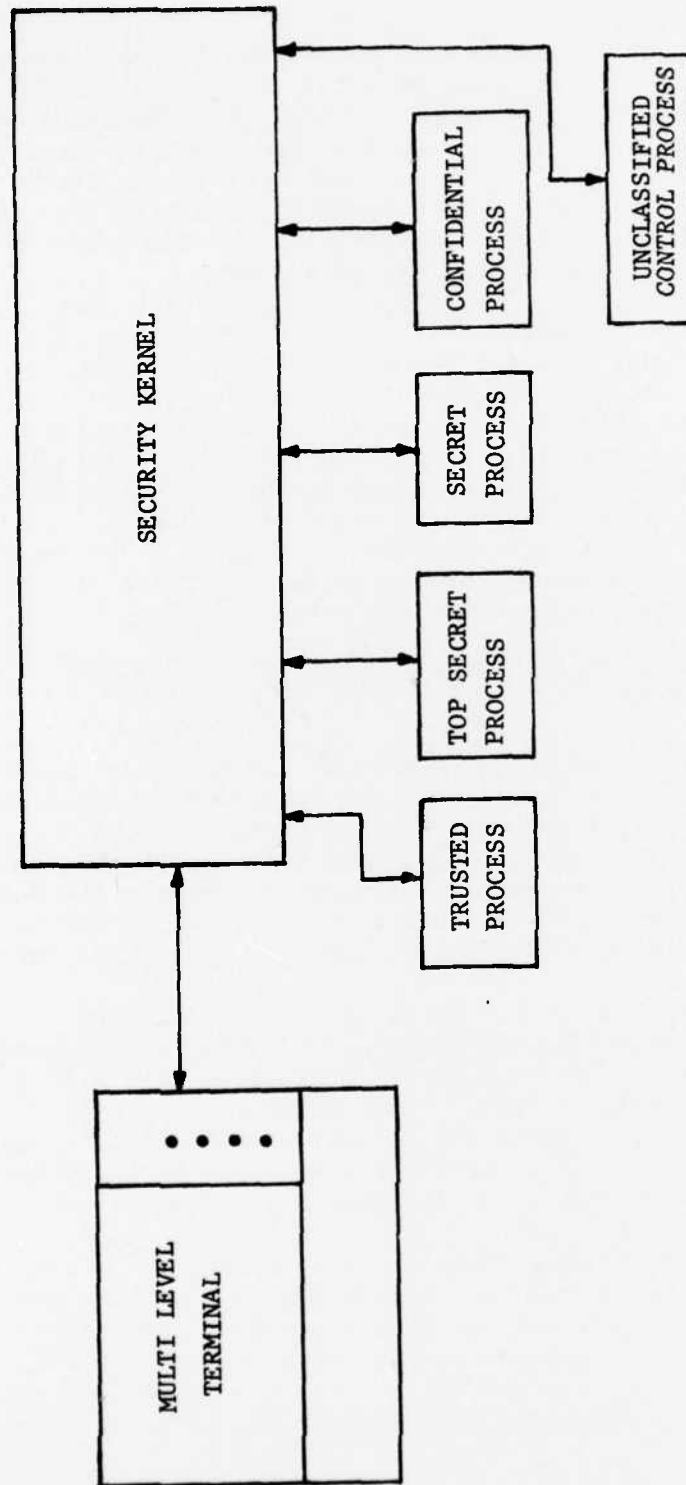


Figure 1. Sigma Process Structure

## MULTILEVEL TERMINAL

We designed the terminal, used by SIGMA, to enable the user to interact with data at more than one security level at a time. The screen of this "multilevel terminal" is divided into "windows" (see Figure 2), each of which is logically an independent terminal. Each window scrolls independently and may have a different security level. Windows are further divided into domains that have various attributes (e.g., enterable, editable, underlined, etc.). The domain's security level is the same as that of the window.

To keep the user apprised of the level of information he is viewing and entering, we added two sets of lights to the terminal. Each set consists of four lights (one for each security level); one and only one light of each set is on at any time. The first set is mounted on the keyboard; it specifies the classification of the window in which the cursor currently resides. If the user wishes to know the level of any particular piece of information on the screen, he may move his cursor to the information. The second set of lights is mounted next to the screen and specifies the maximum level of information displayed on the screen.

## FORM OF COMMAND INPUT

We designed command input so that it could be done through a separate window that is normally at the unclassified level in order to keep the majority of the user's state information at unclassified. Certain commands, such as the "find text string" command, have potentially classified arguments. For these commands the security level of the command window is raised to the level of the object that the command is affecting before the user enters the parameters.

Strict enforcement of the security model eliminates any possibility of a security compromise: a write-down path through the system that could be used to release information of a higher security level to a lower security level. [15] However, even with the enforcement of the model, there are several situations in a message system where the user, by following instructions given by the system, can inadvertently compromise small amounts of information.

Consider the following example: A user asks for a list of all his messages with a subject having word "x" in them. To perform this operation, the user must be at the security level of the file that he is looking at--greater than or equal to the security level of all the messages within that file. The enforcement of the \*-property forces the result of this examination to be at the level

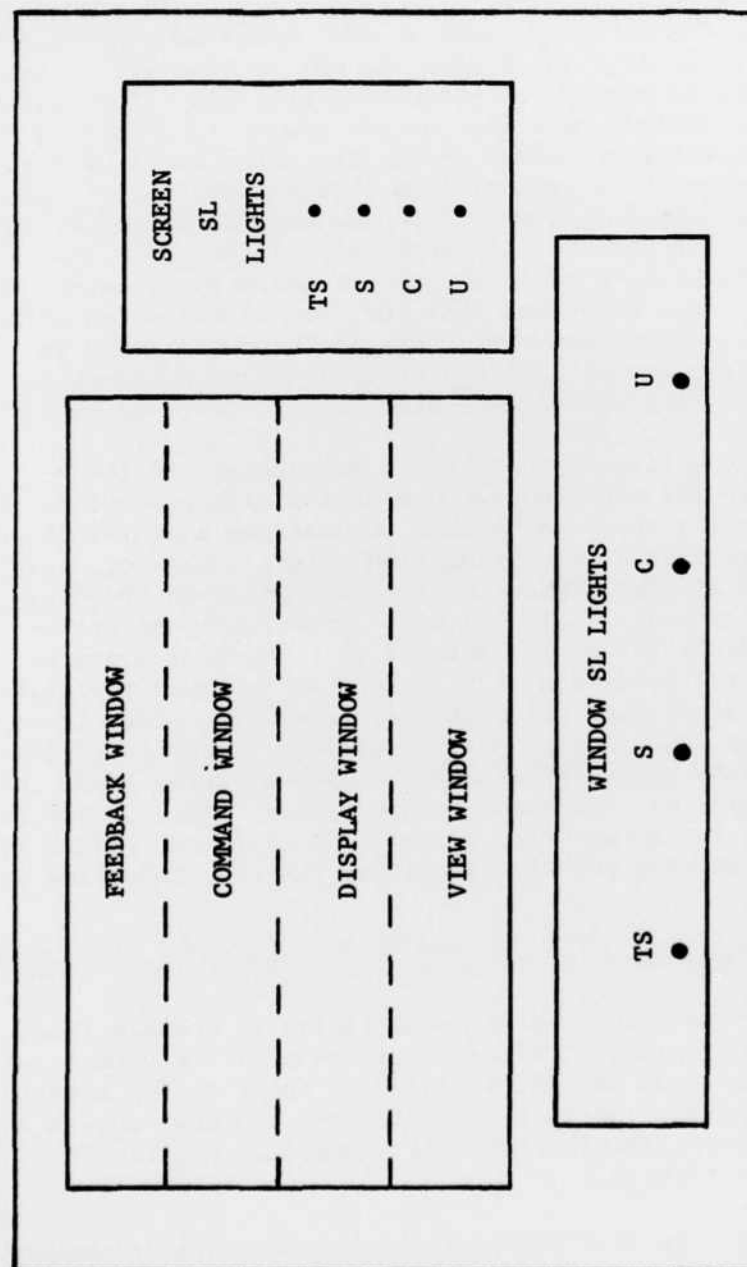


Figure 2. Multi Level Terminal

at which the examination was performed--the security level of the file. Should the user then decide to perform any modification to a message returned by this examination that has a security level lower than the file security level, the \*-property would require him to: issue commands at the security level of the message that he desired to modify, and tell the system the unique identification of the message told to him by the classified process. (The unique identification is required here because the system is unable to pass the desired identifier "down" due to the enforcement of the \*-property.) However, this transmission through the user of the message-id from the higher process to the lower process is, itself, a violation of the \*-property. Although it is conceivable that a maliciously written program could use this \*-property violation to compromise information, we assume that the user serves as an effective filter in this write-down path (both in "bandwidth" and in checking for reasonableness), thereby precluding any reasonable software means of making use of this path.

Because of the hardships that strict enforcement of the \*-property imposes on the user and because of the existence of \*-property violations, a case can be made to ease the user interface in situations where this type of violation exists. The improvement takes the form of allowing SIGMA, in violation of the \*-property rule of the security model, but with user concurrence, to write-down the unique identifier of the message that the user wants to modify. We limit the bandwidth of this type of \*-property violation, so that it is no larger than the path that otherwise occurs through actions of the user, by allowing only a specific amount of fixed-format information to be transmitted to a lower security level and then only if the user has depressed an appropriate function key that is linked directly to the security kernel. Allowing the system to transmit this information greatly simplifies the user interface.

#### TRUSTED PROCESS FUNCTIONS

Certain functions need special capabilities to operate (such as the passing of message identifiers) but are relatively message-system dependent and thus are not included in the security kernel. We group these functions together in a "trusted process" that has the ability to transfer information in a controlled fashion in violation of the \*-property.

The trusted process in SIGMA performs four functions: change classification; message release; command completion signals; and entry list transmission.

### Change Classification

SIGMA allows users to change the classification of text that they are allowed to access. When this happens, the trusted process clears the screen and presents the text in a simple fashion (19 lines at a time) for confirmation. When the user has confirmed the entire object, the trusted process logs this action and passes the text to a process at the new security level for refiling.

### Message Release

In the military, formal messages are released with the commander's signature, or the signature of his designee. Therefore, we consider the act of releasing a message a security event. To control message release, we require that the trusted process insure that the user requesting the release is authorized to release and that this user is making the request from an authorized terminal.

An additional security consideration with message release is that some AUTODIN terminals (ours in particular) treat the message header as unclassified. In SIGMA this header is created in the same window as the message text. Therefore, releasing a message implicitly lowers the classification of the header information. During message release the trusted process requires the user to confirm that all of the header information is actually unclassified. The trusted process logs this action before releasing the message.

### Command Completion Signals

We have based the SIGMA design on the concept of an unclassified process that receives the majority of the commands, determines the proper security level needed to execute these commands, and then activates a process at that security level to perform the execution. The disadvantage of this approach is that, should an error occur between the unclassified control process and the classified operational process, the classified process cannot ask for clarification. Thus error recovery is difficult. This problem is referred to as the open loop problem.

Presently we believe that the best solution to the open loop problem is to allow the trusted process to close the loop when an error of this type is encountered, provided the user has depressed a function key since the last such request. Closing the loop improves recovery but has an impact on security, since a \*-property violation exists when the loop is closed. As in command input, requiring a user action between successive writedowns restricts the bandwidth of this operation.

### Entry Lists

When a user process needs to write down a list of message identifiers, "entry lists", it passes this list to the trusted process for user confirmation directly. The trusted process checks the format and bounds of the entry numbers, and asks the user to directly confirm the number of entries being processed at each security level. Thus, the user has the ability to directly monitor (and control) the bandwidth of the writedown channel. This separate step is reasonable even from the user interface side, for if the number is too high or too low, the user can see that he specified his request incorrectly. (4)

---

(4) If the entry list has only one element, then the appropriate function key is sufficient and the further "confirm 1 entry" step is omitted. This operation allows the user to point to a single entry or mention it by number or context (CURRENT, NEXT) for a display, reply, file, etc., without being required to do more than use the proper function key to enter or confirm the command.

## SECTION VI

### KERNEL REQUIREMENTS

In order to be able to support the SIGMA architecture, the security kernel must provide certain additional features not found in kernels designed to date, including a terminal multiplexor for the multilevel terminal, a variety of object sizes, the ability to support large numbers of processes, an efficient inter-process communication facility, and a policy that can support "trusted" processes.

#### MULTILEVEL TERMINAL CERTIFICATION

Since the terminal supports the simultaneous display and editing of data at different classifications, we must demonstrate that the terminal 1) maintains the proper levels for all information it contains (possibly 20,000 characters) and 2) marks all information returned to the computer with the proper security level. It is the terminal's responsibility to assure that no information entered in a window by either the user (doing local editing) or the application computer is transferred to any other window. While at first pass the certification of the terminal may seem trivial, one must consider that the terminal code is currently produced for a single INTEL 8080 and occupies 32K bytes of PROM. Eventually multiple 8080's, application of Denning's flow control [16], or the introduction of a kernel in the terminal will be necessary to guarantee separation of the windows.

#### TERMINAL MULTIPLEXOR

A significant problem is the method for attaching the multilevel terminal to a secure system. We have identified two alternatives: each window could have a unique connection to the system or the kernel could multiplex all information to a terminal over the same communication line. We have chosen the multiplexing approach in order to minimize the number of terminal lines.

Communication between the system and the terminal is in the form of NOTICES and DISPATCHES.<sup>(5)</sup> The terminal multiplexor must assure that each NOTICE received from the terminal is directed to a user process whose security level is the same as the security level of the window. The multiplexor must also forward Function Key Notices to the trusted process to provide for the capability for a controlled write-down of message identifiers.

The terminal multiplexor must insure the correctness of all DISPATCHES to the terminal. With the exception of "window allocation" DISPATCHES, the terminal multiplexor need only check the window identifier and length to assure that the user process is communicating with a window to which it has access. All requests for terminal window allocations and deallocations must be done by the unclassified user control process. This process provides the terminal multiplexor with the security level for all newly created windows. The unclassified process can, at a later time, request a change in a window classification by notifying the multiplexor. If this new security level is lower than the current window security level, the multiplexor must erase the information currently in the window.

#### PROCESS STRUCTURE

The design of the kernel's process structure will have significant implications for the performance of SIGMA. On traditional timesharing systems, such as TENEX, process creation is expensive, and process swapping is lengthy. In order for SIGMA to operate efficiently the kernel must be able to 1) support large number of processes; 2) allow for fast process creation and deletion; and 3) swap processes with little overhead. Large numbers of processes are required because SIGMA requires several active processes per user. If SIGMA is extended to handle compartmented intelligence, fast process creation and deletion would be required. Finally, because large numbers of processes are doing small amounts of processing, process swapping occurs often. To expect a kernel to provide this type of support may require significant hardware support.

---

<sup>(5)</sup> NOTICES and DISPATCHES are packets of information to and from the host computer respectively.

## INTERPROCESS COMMUNICATION

Equally important to the efficient operation of SIGMA are the speed and types of interprocess communication provided for by the kernel. SIGMA will require the kernel to support both preemptive (interrupt-like) and non-preemptive (message-like) types of interprocess communication. In addition, the latter mechanisms must support small messages for passing message-ids and large messages for transmitting entire command strings.

## FILE SYSTEM

The file system is often one of the most complex portions of the kernel, a quality which can cause unnecessary overhead. For example, SIGMA does not require the kernel to provide a file organization such as a directory hierarchy. A "flat file" system is entirely adequate and can be used more efficiently. The only special requirements for SIGMA are that the kernel should support both small files (512 bytes) and large files (10K bytes).

## SYSTEM INTEGRITY CONTROLS

To support SIGMA the security kernel must provide a mechanism that implements the notion of least privilege. This mechanism has been given the name "System Integrity" [17]. SIGMA uses three separate privileges: a secure write-down privilege used by the trusted process to reclassify text; a release privilege used to restrict the releasing of messages to a select group; and a system security officer privilege used to initiate and set the security level of new users.

The primary rule that the system integrity control must obey is: to modify an object or execute a kernel call a subject's system integrity level must be greater than or equal to the system integrity level of the object or kernel call. [18] There are no rules on reading or executing programs (programs in execution use the system integrity level of the process that they are executing under). We must therefore demonstrate that each subsystem with a system integrity level greater than system low (non-kernel, no privileges) does not execute any programs other than ones that we know execute properly.

## SECTION VII

### SUMMARY

The design of SIGMA demonstrates that it is possible to build a secure message processing system based on the kernel approach to multilevel security. We have shown the refinements to the approach that are needed to achieve a usable interface and have documented the features that a security kernel must provide to support a secure message processing system efficiently. The techniques used in designing SIGMA should be directly applicable to other transaction-oriented or data base management systems.

## LIST OF REFERENCES

1. Bolt, Beranek and Newman Inc., "HERMES Message System Information Package: MME-HERMES an Introduction," draft in preparation, Bolt, Beranek and Newman Inc., February 1977.
2. A. Vezza, M. S. Broos, "An Electronic Message System: Where Does It Fit?" Trends and Applications 1976: Computer Networks, Gaithersburg, Maryland, November 1976, pp. 89, 97.
3. "ESD 1974 Computer Security Developments Summary," MCI-75-1, Electronic Systems Division (AFSC), L. G. Hanscom Field, Bedford, Massachusetts, December 1974.
4. J. P. Anderson, "Computer Security Technology Planning Study," ESD-TR-73-51, Volume I and II, James P. Anderson & Co., Fort Washington, Pennsylvania, October 1972.
5. D. E. Bell and L. J. LaPadula, "Secure Computer Systems: Mathematical Foundations and Model," M74-244, The MITRE Corporation, Bedford, Massachusetts, October 1974.
6. K. G. Walter, W. F. Ogden, W. C. Rounds, F. T. Bradshaw, S. R. Ames, Jr., and D. G. Shumway, "Primitive Models for Computer Security," ESD-TR-74-117, Case Western Reserve University, Cleveland, Ohio, January 1974.
7. D. E. Bell and E. L. Burke, "A Software Validation Technique for Certification, Part 1: The Methodology," ESD-TR-75-54, Volume I, The MITRE Corporation, Bedford, Massachusetts, April 1975 (AD 009849).
8. S. R. Ames, Jr., J. K. Millen, "Interface Verification for a Security Kernel," System Reliability and Integrity, Volume 2: Invited Papers, Infotech International, 1978, pp. 1-21.
9. D. E. Bell and L. J. LaPadula, "Secure Computer System," ESD-TR-73-278, Volume I-III, The MITRE Corporation, Bedford, Massachusetts, November 1973-June 1974.

10. R. M. Graham, "Protection in an Information Processing Utility," Communications of the ACM, Volume 11, Number 5, May 1968, pp. 365-369.
11. J. K. Millen, "Security Kernel Validation in Practice," Communications of the ACM, Volume 19, Number 5, May 1976, pp. 243-250.
12. L. Robinson, P. G. Neumann, K. N. Levitt, and A. R. Saxena, "On Attaining Reliable Software for a Secure Operating System," 1975 International Conference on Reliable Software, Los Angeles, California, April 1975, pp. 267-284.
13. S. R. Ames, "A Security Compliance Study of the Air Force Data Services Center Multics System," MTR-3065, The MITRE Corporation, June 1975.
14. S. R. Ames, Jr., "User Interface Multilevel Security Issues in a Transaction-Oriented Data Base Management System," MTP-178, The MITRE Corporation, Bedford, Massachusetts, December 1976.
15. S. B. Lipner, "Comment on the Confinement Problem," ACM Operating Systems Review, Volume 9, Number 5, May 1975, pp. 192-196.
16. D. E. Denning and P. J. Denning, "Certification of Programs for Secure Information Flow," Communications of the ACM, Volume 20, Number 7, July 1977, pp. 504-513.
17. S. R. Ames and W. W. Plummer, "TENEX Security Enhancements," MTR-3217, The MITRE Corporation, Bedford, Massachusetts and Bolt, Beranek and Newman, Inc., Cambridge, Massachusetts, April 1976.
18. K. J. Biba, "Integrity Considerations for Secure Computer Systems," ESD-TR-76-372, The MITRE Corporation, Bedford, Massachusetts, December 1976.

DISTRIBUTION LIST

INTERNAL

D-70

J. J. Croke  
W. S. Melahn  
J. W. Shay

D-73

S. B. Lipner

D-74

R. S. Gardella

D-75

S. R. Ames (10)  
E. H. Bensley  
E. L. Burke (5)  
J. A. Clapp  
M. Ferdman  
F. C. Furtek  
A. G. Gann  
N. C. Goodwin  
S. W. Hosmer  
J. L. Mack  
J. K. Millen  
D. G. Miller  
G. H. Nibaldi  
M. A. Padlipsky  
C. M. Sheehan  
S. L. Smith  
S. A. Swernofsky  
J. D. Tangney  
P. S. Tasker (5)  
B. N. Wagner  
P. T. Withington  
J. P. L. Woodward

W-31

S. Bergman  
S. I. Schaen  
D. C. Wood

W-37

T. J. Leso

EXTERNAL

Defense Advanced Research Projects  
Agency  
Information Processing Techniques  
Office  
1400 Wilson Blvd.  
Arlington, Virginia 22209

D. Adams (10)  
W. Carlson (10)

Naval Research Laboratory  
45 15 Overlook Ave., S.W.  
Washington D.C.

S. Wilson (2)